Naturalness of Attention: 
Revisiting Attention in Code Language Models

Mootez Saad and Tushar Sharma
{mootez,tushar}@dal.ca
Dalhousie University
Halifax, Nova Scotia, Canada

ABSTRACT
Language models for code such as CodeBERT offer the capability to learn advanced source code representation, but their opacity poses barriers to understanding of captured properties. Recent attention analysis studies provide initial interpretability insights by focusing solely on attention weights rather than considering the wider context modeling of Transformers. This study aims to shed some light on the previously ignored factors of the attention mechanism beyond the attention weights. We conduct an initial empirical study analyzing both attention distributions and transformed representations in CodeBERT. Across two programming languages, Java and Python, we find that the scaled transformation norms of the input better capture syntactic structure compared to attention weights alone. Our analysis reveals characterization of how CodeBERT embeds syntactic code properties. The findings demonstrate the importance of incorporating factors beyond just attention weights for rigorously understanding neural code models. This lays the groundwork for developing more interpretable models and effective uses of attention mechanisms in program analysis.

KEYWORDS
Attention Analysis, Language Models of Code, Norm Analysis, Interpretability

ACM Reference Format:

1 INTRODUCTION

Obtaining effective representations of source code is crucial for many program analysis tasks such as code search, code completion, and program translation. Earlier code representation approaches, such as Code2Vec [2] and Code2Seq [1] demonstrated initial progress in learning distributed vector representations of code. However, these methods are limited in their modeling capacity and do not fully capture the rich semantics of code. To overcome the limitations in early code representation techniques, Transformer-based [16] neural models have emerged as a promising paradigm for learning effective code representations. Inspired by their success in natural language processing, architectures such as BERT [4] have been adapted to code representation, leading to the emergence of Language Models of Code (LMC), such as CodeBERT [5], GraphCodeBERT [7], CodeT5 [19], and Code Llama [13]. The self-attention mechanism powering Transformers provides stronger representation capabilities compared to earlier architectures such as Recurrent Neural Networks (RNNs). This has enabled Transformer-based models to establish new state-of-the-art results across a variety of software engineering tasks that involve source code analysis, processing, and manipulation by learning better semantic representations of programs.

Despite such advances, a major limitation of LMCs is their black box nature and lack of interpretability. While models such as CodeBERT show impressive performance on downstream tasks, it is unclear which properties of code they capture or learn internally.

To address these interpretability issues, an emerging area of research involves analyzing and probing these complex neural networks through attention visualization and representation analysis. Sharma et al. [15] found BERT models trained on code exhibit key attention differences from natural language - namely, higher focus on identifiers over special tokens like [CLS] and more localized context. Wan et al. [18] investigated the encoded syntactic patterns within the attention distributions of CodeBERT and GraphCodeBERT. However, in the realm of natural language, Kobayashi et al. [10] note that attention weights alone may not reveal the full perspective of the patterns learned by the model. Based on the naturalness property of software [8], techniques effective for analyzing natural language models may also lend insight into source code models. This motivates revisiting attention-based analysis by incorporating the factors that were previously ignored when analyzing LMCs.

In this paper, we revisit the mathematical formulation of the Multithead Attention (mha) [10] to illustrate how it is composed of two factors: attention weights and the transformation of input. Given this new reformulation, we perform a trend analysis of the Transformer layers of CodeBERT on Java and Python to study the differences between these two factors. We show how including the previously ignored effect leads to a better alignment with the syntactic properties of source code compared to the attention weights.

We make the following contributions:
- First, we perform a layer-wise analysis to study the trends of attention weights and the transformation of input for an LMC (i.e., CodeBERT). To the best of our knowledge, this is the first study that emphasizes the need to consider the transformation of input along with attention weights in the context of an LMC.
- Second, we compare the capacity of the two factors to capture the syntactic properties of source code.

We make the replication package, including code and data, available online [14].
2 BACKGROUND AND MOTIVATION

The core component of the Transformer architecture is the Multitheaded Attention (MHA), which is composed of multiple Self-Attention (SA) heads. For instance, in CodeBERT, which is based on RoBERTa’s [11] architecture, each MHA layer is composed of 12 SA heads. Let $h$ denote the number of heads, and $X$ be the input sequence of length $n$, where each token is embedded in $\mathbb{R}^d$ ($d^2 = 64$ in CodeBERT). In each head, a sequence is projected into three matrices: Query ($Q_{n \times d^2}$), Key ($K_{n \times d^2}$) and Value ($V_{n \times d^2}$). Formally, these matrices are defined as follows:

For $i \in [1 \ldots h]$

$$Q^{(i)} = X \cdot W^Q_{O}, \quad K^{(i)} = X \cdot W^K_{O}, \quad V^{(i)} = X \cdot W^V_{O}$$

(1)

The attention matrix $A$ is computed by applying the $\text{softmax}(\cdot)$ function on the result of the multiplication of the Query and Key matrices, scaled by the square root of their dimension $d'$.

$$A^{(i)} = \text{softmax}\left(\frac{Q^{(i)T}K^{(i)}}{\sqrt{d'}}\right)$$

(2)

Then, the attention matrix is multiplied by the Value matrix to obtain the attention output $z$.

$$z^{(i)} = A^{(i)} \cdot V^{(i)}$$

(3)

Finally, concatenating the output of each head and multiplying it by a weight matrix $W_{O}^{d' \times hd'}$, gives the output of the MHA layer.

$$Z_{\text{MHA}} = [z_1; \ldots ; z_h]_{n \times hd'},$$

(4)

$$Y_{\text{MHA}} = Z \cdot W_{O}$$

(5)

$Y_{\text{MHA}}$ can be reformulated given the linearity of matrix multiplication. To build the intuition, let us consider the calculation of the entry located at the $1^{\text{st}}$-row and $1^{\text{st}}$-column of $Y_{\text{MHA}}$. It is done by taking the dot product of the $1^{\text{st}}$-row of $Z_{\text{MHA}}$ and the $1^{\text{st}}$-column of $W_{O}$.

$$Y_{\text{MHA}}[1, 1] = \sum_{i=1}^{H} Z_{\text{MHA}}[1, i]W_{O}[i, 1]$$

(6)

We can decompose Equation (6) into $h$ summations,

$$Y_{\text{MHA}}[1, 1] = \sum_{i=1}^{d'} Z_{\text{MHA}}[1, i]W_{O}[i, 1] + \ldots + \sum_{i=hd' - d' + 1}^{hd'} Z_{\text{MHA}}[1, i]W_{O}[i, 1]$$

(6)

By extension and with reference to Figure 1, we can express $Y_{\text{MHA}}$ as the sum of $h$ matrices calculated from the multiplication of the submatrices from $Z_{\text{MHA}}$ and $W_{O}$. This entails that Equation (5) can be rewritten as follow,

$$Y_{\text{MHA}} = Z_{\text{MHA}} \cdot W_{O} = \sum_{i=1}^{h} z^{(i)} \cdot W_{O}^{(i)}$$

(9)

and if we plug in Equation (3) and Equation (1) we obtain,

$$Y_{\text{MHA}} = \sum_{i=1}^{H} A^{(i)} \cdot V^{(i)} \cdot W_{O}^{(i)}$$

(10)

(10)

We can see from this reformulation that the information $Y_{\text{MHA}}$ holds is the result of the contribution from two factors: the attention weights $A$ and the transformation $f(.)$ applied on the input $X$. A token $t_i \in X$ can have a high attention weight $a_i > 0$ and, at the same time, a low contribution from its transformation $||f(t_i)|| \approx 0$ [10]. This implies that the properties deduced from the analysis of $MHA$ cannot solely be attributed to the attention weights. The previous studies that performed attention analysis of Language Models of Code (LMC) concentrated on probing the attention weights to see the type of patterns they exhibit and how well they align with the properties of source code [15, 18].

Motivated by this argument, we would like to revisit the attention analysis of LMC by considering the missing factor, i.e., the scaled transformation $||a f(x)||$.

The primary objective is to thoroughly comprehend the attention mechanism’s properties when applied to source code. To this end, we conduct a preliminary empirical study and answer the following research questions:

**RQ1:** How do the general trends across layers between attention weights $a$ and the scaled transformations norms $||a f(x)||$ compare?

**RQ2:** How does $||a f(x)||$ align with the syntactic structure of source code compared to attention weights?

3 EXPERIMENTS AND RESULTS

In this section, we will present our methodology and the conducted experiments to answer the research questions stated above.

3.1 Methodology

**Models:** In this study, we considered CodeBERT [5], a pretrained language model of code that adopts the architecture and pertaining strategy as RoBERTa [11]. We chose such a model to follow Wan et al. [18]. In addition, it is one of the earliest LMCs which has spawned many follow-up works. Analyzing it provides a strong baseline for future comparative studies on other related models.

It consists of 12 Transformer [16] layers, each encompassing 12 self-attention heads. It was trained on a set of bimodal instances (i.e., pairs of natural language and programming languages), across six programming languages from the CodeSearchNet dataset [9].
Naturalness of Attention: Revisiting Attention in Code Language Models

Data: We used CodeSearchNet dataset [9] to create corpora for two programming languages: Java and Python. Each corpus consists of 5,000 randomly sampled code snippets with lengths less than 512 tokens.

3.2 RQ1—Trend Analysis

Through this research question, we aim to investigate the trends, at a macro level, in the behaviours of the attention weights $\alpha$ versus the scaled norms $\| \alpha f(x) \|$ across the layers of CodeBERT.

**Approach:** From each self-attention head, we extracted the attention weights $\alpha$, transformation norm $\| f(x) \|$ and scaled transformation norm $\| \alpha f(x) \|$ matrices, for each instance. Since CodeBERT was trained using a WordPiece tokenizer [20], each word can be tokenized further into subtokens. Given that our analysis was carried out at a word level, we follow the same procedure done by Clark et al. [3] and convert token-token maps to word-word maps by taking the average of a word’s subtokens. To make our analysis granular, we group tokens by their categories: Keywords, Identifiers, Literals, Operators and Special Symbols. Each category is defined according to each programming language grammar specification (Java [12] and Python [6]).

**Results:** Figure 2 depicts the variation of the average attention weights and the average scaled transformation norms across each layer for each token category in both datasets. Aligned with the findings of Kobayashi et al. [10] and Clark et al. [3], and in contrast to the results of Sharma et al. [15], the special tokens displayed higher average attention compared to other token types. Specifically, <s> had the highest average attention between Layers 2 and 4, which then decreased until Layer 7. Then, its attributed attention kept on increasing until Layer 10. Interestingly, the drop in <s>’s attention between Layers 5 and 7 appeared to transfer to </s>, whose average attention peaked within this range. This pattern held across programming languages, with similar trends in both Java and Python corpora, though minor differences arose in precise attention values. For instance, <s> attention remained constant between Layers 2 and 3 for Java, whereas it slightly declined in the Python dataset.

However, this pattern is different when calculating the scaled transformation norms $\| \alpha f(x) \|$. The contribution of these tokens is lower compared to other categories such as Identifiers and Special Symbols. This indicates, that similar to BERT, when CodeBERT does not find information in the input, it assigns higher attention values to these tokens given that the attention weights should sum up to 1 (due to the $\text{softmax}(\cdot)$ function).

Although the ranking of each token category at each layer appears to be consistent between $\alpha$ and $\| \alpha f(x) \|$, there is some contrast between the patterns observed at each layer. For example, if we look at the trend of attention weights of the **Keyword** tokens in the Python dataset in Figure 2a, we see that the attention values drop between Layer 1 and Layer 2 and remain relatively constant between Layers 2 and 4. In contrast, the values of $\| \alpha f(x) \|$ between Layers 1 and 3 are increasing for this category. This contrast effect is also observed for other types such as **Identifiers**. Generally, in both datasets, we see that in some layers, when the attention weights are constant (e.g., L2-L4 and L5-L8) the scaled transformation norms exhibit either a peak or a decline. One explanation can be the **cancelling effect** of $\alpha$ and $\| f(x) \|$ that was mentioned in Section 2, hence, the contrast between $\alpha$ and $\| \alpha f(x) \|$. Figure 3 further illustrates this cancelling effect for the special tokens <s> and </s>, and the **Literals** category.

**Summary:** The results show that the behavior of attention weights and the scaled transformation norms $\| \alpha f(x) \|$ differ significantly. The two components of Multiheaded Attention i.e., $\alpha$ and $\| f(x) \|$, often exhibit a cancellation effect. Such contrast entails that including other variables, i.e., the transformation norm $\| f(x) \|$, when performing attention analysis, might lead us to more comprehensive and explainable results. In other words, extending the analysis to regions other than attention weights might reveal additional insights about the language model’s capacity to model the relations and properties of source code.

3.3 RQ2—Syntactic Alignment

In this section, we analyze the syntactic properties that are embedded in $\| \alpha f(x) \|$ compared to those in the attention weights.

**Approach:** Vig et al. [17] proposed a metric, $p_{\alpha}(g)$, that measures the agreement between an attention map (i.e., the attention matrix or weights) and a property map generated by an indicator function $g$. The function $g(i, j) = 1$ if a given property exists between two tokens $i$ and $j$, 0 otherwise. Wang et al. [18] defined $g$ to return 1 if the pair $(i, j)$ share the same parent in the Abstract Syntax Tree.
where \((\ast)\) of a code snippet \(x\). Their intuition was that attention defines the closeness of each pair of code tokens. This score is formally defined in Equation (11),
\[
p_a(g) = \frac{\sum_{x} \sum_{i=1}^{n} \sum_{j=1}^{n} f(i, j) \cdot 1_{a_{i,j} > \theta}}{\sum_{x} \sum_{i=1}^{n} \sum_{j=1}^{n} 1_{a_{i,j} > \theta}}
\]
where \(1_{a_{i,j} > \theta}\) is an indicator function that selects high-confidence attention weights \((\theta = 0.03\text{ in} [17, 18])\). In other words, \(1_{a_{i,j} > \theta}\) evaluates to 1 if \(a_{i,j} > \theta\), and 0 otherwise. Equation (11) sums over all token pairs \((i, j)\) where the attention \(a_{i,j} > \theta\). It counts how many of these high-confidence attention pairs connect tokens that are syntactically related according to the AST \((i.e., f(i, j) = 1)\) over the dataset. Dividing this count by the total number of high-confidence pairs gives the proportion of attention connections that align with the AST structure. This proportion indicates how well the attention matches syntactic relationships.

\[\text{Figure 3: Attention } (a) \text{ and transformation norm maps } (||f(x)||) \text{ for } <s>, </s>, \text{ and } \text{Literals} \text{ from the Java dataset. For each type of token, the left and right figures refer to the attention and norm map respectively. Regions that show the contrast relation are highlighted with the same colour.}\]

(\text{4 CONCLUSIONS AND FUTURE WORK})

In this work, we have revisited the mathematical definition of \text{MHA} from prior works in natural language. We showed how the attention mechanism is not merely composed of the attention weights. The preliminary findings indicate that incorporating scaled transformation norms provides new perspectives on what code properties are captured by attention.

The presented work can be extended in a variety of directions. The first extension point is to investigate how these findings could vary across other programming languages and models. Applying the same methodology to models such as GraphCodeBERT [7] and CodeT5 [19] and other languages such as JavaScript and Go will test the generalizability of our results. Along similar lines, we aim to evaluate models trained with techniques that are more programming language-oriented. CodeBERT, despite being trained on NL-PL pairs, was pre-trained with the same objective as RoBERTa to model natural language. On the other hand, GraphCodeBERT encodes data flow paths in its input, which captures more source code properties other than its sequential nature. Similarly, CodeT5 uses an identifier-aware pre-training task that allows the model to determine which code tokens are identifiers and to recover them when they are masked. Determining if specialized training objectives modify attention behaviors will clarify how different representations are learned. The goal is to connect training procedures with resultant attention patterns.
REFERENCES


